

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

TITLE OF THE INVENTION

**INHERITANCE OF OBJECT'S PROPERTIES IN AND OUT OF DIFFERENT APPLICATION
CONTEXTS IN PROPERTIES FILE OBJECTS**

INVENTORS

**RONI KORENSHTEIN
BENNETT TING**

Prepared by

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(303) 740-1980

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: E: 845313685 US
Date of Deposit: April 4, 2001
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service
"Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has
been addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231

Fran C. Rolfsen

(Typed or printed name of person mailing paper or fee)

Fran C. Rolfsen

(Signature of person mailing paper or fee)

4-6-01

(Date signed)

**INHERITANCE OF OBJECT'S PROPERTIES IN AND OUT OF DIFFERENT APPLICATION
CONTEXTS IN PROPERTIES FILE OBJECTS**

[0001] This application claims the benefit of U.S. Provisional Application Nos.

5 60/198,718 and 60/205,700, filed April 20, 2000 and May 19, 2000, respectively, both of which are incorporated herein by reference.

COPYRIGHT NOTICE

[0002] Contained herein is material that is subject to copyright protection. The

10 copyright owner has no objection to the facsimile reproduction of the patent disclosure by any person, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all rights to the copyright whatsoever.

BACKGROUND OF THE INVENTION

15 Field of the Invention

[0003] The invention relates generally to apparatus and methods for accessing and retrieving attribute name value pairs from properties data stores, such as properties files.

More particularly, the invention relates to syntax enhancements and mechanisms for extending existing flat properties data stores to allow definition and access of properties
20 on an object-by-object basis, allow implementation of inheritance of properties through some hierarchy of objects, and allow specification and overriding of properties of objects as they are used in different contexts.

Description of the Related Art

[0004] Java has a special built in mechanism for handling internationalization. One use of this mechanism encourages putting locale specific properties in “properties files” (e.g., .properties files). This file contains attribute name value pairs. The normal syntax of the properties files are lines containing:

<attribute_name>=<value>

[0005] The “=” separates the key from the value in the properties file. The existing Java properties file mechanism allows an application to request the value of a specific attribute from a specific properties file and Java returns the value of that attribute as specified in the named file. This is done by loading a specified properties file into an object of type *ResourceBundle* and then querying a specified attribute. The following example illustrates this. Suppose the file *details.properties* contains the following two lines:

nameString=name

prompt=please enter your full address

The application, using the Java application programming interface (API), can query the value of the “*prompt*” property in the properties file named “*details.properties*” and get the result “*please enter your full address*”. This is how it is done:

ResourceBundle resourceBundle = ResourceBundle.getBundle(“details”);

String promptValue = ResourceBundle.getString(“prompt”);

[0006] Several limitations of existing properties files and their respective syntaxes have become apparent to the assignee of the present invention. First, in the case of Java, the properties file and the resource bundle are both flat, non-object oriented collections of key-value (attribute-value) pairs. Consequently, attributes may not be specified on an object-by-object basis. Second, there is no way to perform inheritance. Third, no mechanism is provided to specify and/or override properties of objects as they are used in different contexts. As a result, programmers do not have the ability to define default values of objects that may be customized or overridden when the objects are running in a particular context. These and other limitations of the Java properties file syntax and usage model, make the specification of attributes both cumbersome and time consuming.

BRIEF SUMMARY OF THE INVENTION

[0007] Apparatus and methods are described for accessing and retrieving attribute

name value pairs from properties data stores, such as properties files. According to one

embodiment, attribute values may be associated with properties data store objects. First,

5 an attempt is made to determine a value of an attribute of an object by formulating and submitting a first query against the properties data store based upon a combination of a current context identifier, an object identifier associated with the object, and an attribute identifier associated with the attribute. If the first query is unsuccessful at locating the value, then the value is determined by formulating and submitting a second query against
10 the properties data store based upon a combination of the object identifier and the attribute identifier.

[0008] Other features of the present invention will be apparent from the accompanying drawings and from the detailed description that follows.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0009] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0010] **Figure 1** is an example of a client-server system in which one embodiment of the present invention may be implemented.

[0011] **Figure 2** is an example of a computer system upon which one embodiment of the present invention may be implemented.

[0012] **Figure 3** is a block diagram of a software architecture for a prerequisite enforcing framework according to one embodiment of the present invention.

[0013] **Figure 4** is a block diagram of a software architecture for an object application according to one embodiment of the present invention.

[0014] **Figure 5A** is a flow diagram illustrating web page sub-component display processing according to one embodiment of the present invention.

[0015] **Figure 5B** is a flow diagram illustrating processing block 530 of **Figure 5A** according to one embodiment of the present invention.

[0016] **Figure 6** is a block diagram of a software architecture for an object application according to another embodiment of the present invention.

[0017] **Figure 7A** is a flow diagram illustrating web page display processing according to one embodiment of the present invention.

[0018] **Figure 7B** is a flow diagram illustrating processing block 730 of **Figure 7A** according to one embodiment of the present invention.

[0019] **Figure 8** illustrates prior art properties file syntax.

[0020] **Figure 9** illustrates an improved properties file syntax that enables definition and access of properties on an object-by-object basis according to one embodiment of the present invention.

[0021] **Figure 10** illustrates an improved properties file syntax that enables inheritance of properties according to one embodiment of the present invention.

[0022] **Figure 11** illustrates an improved properties file syntax that enables the specification and/or overriding of object properties as they are used in different contexts according to one embodiment of the present invention.

[0023] **Figure 12** illustrates an improved properties file syntax that enables inheritance of contexts according to one embodiment of the present invention.

[0024] **Figure 13** is a high-level flow diagram that conceptually illustrates property retrieval processing according to one embodiment of the present invention.

[0025] **Figure 14A** is an exemplary context and object hierarchy.

[0026] **Figure 14B** illustrates property retrieval processing according to the object and context hierarchy of **Figure 14A** and the processing of **Figure 13**.

[0027] **Figure 15** is a flow diagram illustrating processing associated with traversal of an object hierarchy and a context hierarchy according to one embodiment of the present invention.

[0028] **Figure 16** is a flow diagram illustrating processing associated with traversal of an object hierarchy without regard to context according to one embodiment of the present invention.

[0029] **Figure 17** is a block diagram the conceptually illustrates software architectural components that enable processing of an improved properties file syntax according to one embodiment of the present invention.

[0030] **Figure 18** illustrates exemplary code for the `getValue` method of **Figure 17** according to one embodiment of the present invention.

[0031] **Figure 19** illustrates exemplary code for the `getWithContextInheritance` method of **Figure 17** according to one embodiment of the present invention.

[0032] **Figure 20** illustrates exemplary code for the `getWithObjInheritance` method of **Figure 17** according to one embodiment of the present invention.

5 [0033] **Figure 21** illustrates exemplary code for the `getWithInheritance` method of **Figure 17** according to one embodiment of the present invention.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

DETAILED DESCRIPTION OF THE INVENTION

[0034] Apparatus and methods are described for accessing and retrieving attribute name value pairs from properties data stores, such as properties files. According to one embodiment, a properties data store defines the relationships between prerequisite and destination pages and/or destination page sub-components. Advantageously, in this manner, the relationships may be modified using an editor and without requiring recompilation.

[0035] In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

[0036] The present invention includes various steps, which will be described below. The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software.

[0037] The present invention may be provided as a computer program product that may include one or more machine-readable mediums having stored thereon instructions, which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions. Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote

computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection). Accordingly, herein, a carrier wave shall be regarded as comprising a machine-readable medium.

5 **[0038]** Importantly, while embodiments of the present invention will be described with reference to web page processing and Java properties files, the apparatus and methods described herein are equally applicable to various other types of user interfaces and mechanisms for maintaining property/attribute information.

10 Network Overview

[0039] The present invention may be included within a client-server system 100 such as that illustrated in **Figure 1**. According to the embodiment depicted in **Figure 1**, a server 130 may be communicatively coupled to clients 110 through a communications medium 120, such as a local area network (LAN) or a wide area network (WAN). A
15 representative data connection between the clients 110 and the server 130 includes a direct-dial, bi-directional data connection, which may be telephone (POTS, i.e., “plain old telephone service”), Integrated Services Digital Network (ISDN), digital subscriber line (DSL) or other similar type of connection. Various other data connections between the clients 110 and the server 130 are possible. For example, the connection may be via a
20 network interface card or via a modem pool (not show) such as that conventionally known in the art for providing access to the Internet and private networks. Alternatively, the connection may be established through an Internet service provider (ISP) that allows the clients 110 to exchange information with the server 130.

[0040] The server 130 includes a novel prerequisite enforcing framework (not
25 shown) that manages, tracks and controls page and/or page sub-component prerequisites. As will be discussed further below, the server 130 may access a properties data store (not shown), such as a Java properties file, that includes information defining the relationships

between prerequisites and destination pages and/or destination page sub-components. Various exemplary software architectures of the prerequisite enforcing framework and preferred syntax and representations of the information in the properties database are described below.

5

Computer System Overview

[0041] A computer system 200 in which features of the present invention may be implemented will now be described with reference to **Figure 2**. Computer system 200 comprises a bus or other communication means 201 for communicating information, and
10 a processing means such as processor 202 coupled with bus 201 for processing information. Computer system 200 further comprises a random access memory (RAM) or other dynamic storage device 204 (referred to as main memory), coupled to bus 201 for storing information and instructions to be executed by processor 202. Main memory 204 also may be used for storing temporary variables or other intermediate information during
15 execution of instructions by processor 202. Computer system 200 also comprises a read only memory (ROM) and/or other static storage device 206 coupled to bus 201 for storing static information and instructions for processor 202.

[0042] A data storage device 207 such as a magnetic disk or optical disc and its corresponding drive may also be coupled to bus 201 for storing information and
20 instructions. Computer system 200 can also be coupled via bus 201 to a display device 221, such as a cathode ray tube (CRT) or Liquid Crystal Display (LCD), for displaying information to a computer user. Typically, an alphanumeric input device 222, including alphanumeric and other keys, may coupled to bus 201 for communicating information and/or command selections to processor 202. Another type of user input device is cursor
25 control 223, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 202 and for controlling cursor movement on display 221.

[0043] A communication device 225 is also coupled to bus 201 for accessing remote servers or other servers via the Internet, for example. The communication device 225 may include a modem, a network interface card, or other well-known interface devices, such as those used for coupling to an Ethernet, token ring, or other types of networks. In any event, in this manner, the computer system 200 may be coupled to a number of clients and/or servers via a conventional network infrastructure, such as a company's Intranet and/or the Internet, for example.

Exemplary Prerequisite Enforcing Framework

[0044] **Figure 3** is a block diagram of a software architecture for a prerequisite enforcing framework according to one embodiment of the present invention. In the embodiment depicted, the software architecture includes a web server 305, an application server 310, and an object application 320. In one embodiment, the web server 305 acts as a HyperText Transfer Protocol (HTTP) interface between the clients 110 and the application server 310.

[0045] The application server 310 appropriately routes requests received from the web server 305 to the object application 320 and conveys output of the object application 320 to the web server 305. For example, in response to page or page subcomponent requests, e.g., Uniform Request Locator (URL) requests, the application server 310 (1) determines the appropriate base agent of the object application 320, i.e., the base agent associated with the requested page or page subcomponent, (2) creates an instance of the base agent, (3) transfers control to the object application 320 by invoking an execute method of the instantiated base agent, and (4) transfers streamed data, received from the object application 320 that has been generated by a page, page subcomponent, page prerequisite, or page subcomponent prerequisite, to the web server 305.

[0046] The object application 320 provides an abstraction for objects, such as containers and components. According to one embodiment, an object may be thought of

conceptually as a hierarchy of potentially nested sub-objects to any number of levels of nesting. The objects and nested sub-objects may be components or containers.

Components are leaf nodes in the hierarchy. Containers are non-leaf nodes in the hierarchy and may contain one or more other containers or components. In the

5 embodiments described herein, containers typically represent pages and components of containers represent the pages' subcomponents, such as charts, tables, scroll lists, data entry forms, etc.

Architecture of an Exemplary Object Application

10 **[0047]** Having briefly described an exemplary environment in which the present invention may be employed, a software architecture of the object application 320, according to one embodiment, will now be described with reference to **Figure 4**. The architecture depicted seeks to facilitate the management, tracking and controlling of page sub-component prerequisites.

15 **[0048]** According to this example, the software architecture of the object application 320 includes a base agent 410, a container object 420, and a sub-component prerequisite factory 430. The base agent 410 includes logic to instantiate and execute the container object 420 that corresponds to a page associated with the base agent 410. In one embodiment, the base agent 410 is instantiated by the application server 310 in
20 response to a URL request for the page associated with the base agent 410. Then, the application server 310 causes the requested page to be displayed by invoking the base agent's execute method. The base agent's execute method, in turn, calls the execute method of the container object 420.

[0049] The container object 420 includes logic to instantiate and execute its sub-
25 objects and determine prerequisites of its sub-objects. The execute method of the container object 420 first determines its sub-objects (e.g., containers and/or components) with reference to a properties data store 440, such as a Java properties file (e.g.,

“<name>.properties”). Then, for each component, the container object 420 determines what the next prerequisite is for the component by consulting with a special sub-component prerequisite factory object, such as the sub-component prerequisite factory 430. As discussed further below, either the next prerequisite or the component will be added to the container object’s list of sub-objects to be displayed. If the container object 420 includes nested containers, when the execute method of the nested containers is invoked, and the nested containers will instantiate and execute their components. When the leaf-level of the hierarchy of objects is reached, those of the components added to the container object’s list of sub-objects to be displayed calculate their individual results and stream out these results to the base agent 410.

[0050] According to one embodiment, the sub-component prerequisite factory 430 includes a method called “getNextPreReq” that returns either the next prerequisite sub-component object for a specified destination sub-component or null. First, the sub-component prerequisite factory 430 determines what, if any, prerequisite sub-component objects have been associated with the specified destination sub-component. In the example depicted, this operation is accomplished by retrieving information from the properties data store 440, such as the value of a “prereq” property. Second, for each of the prerequisite sub-component objects, the sub-component prerequisite factory 430 determines if the requirements for the prerequisite sub-component have been satisfied. As described further below, in one embodiment, this determination is made by calling a “metReqs” method of the prerequisite sub-component object. In any event, if the sub-component prerequisite factory 430 returns a non-null object, it means the destination sub-component is associated with a prerequisite managed by the prerequisite sub-component object that has not yet been satisfied; and rather than displaying the destination sub-component, the prerequisite sub-component object should be displayed to the user. If a null value is returned by the sub-component prerequisite factory 430, it means that either the specified destination sub-component has no prerequisites or that all

the prerequisites for the sub-component have been fulfilled and therefore the destination sub-component may be displayed. Advantageously, according to this novel architecture, the container object 420 is abstracted from the task of prerequisite processing. In alternative embodiments, the container object 420 may be even further abstracted from this process by removing its interface with the properties data store 440 and requiring all access to the properties data store 440 be by way of the sub-component prerequisite factory 430.

[0051] The software architecture of the present example also includes page sub-component objects 450, page sub-component prerequisite objects 460, and a status data store 470. The page sub-component objects 450 are each associated with one or more container objects 420. Importantly, the page sub-component objects 450 and the page sub-component prerequisite objects 460 may be coded independently of each other and do not even need to be aware of each other. That is, the page sub-component objects 450 have no knowledge regarding their prerequisites and include no code relating to determining whether or not such prerequisites have been met. Advantageously, by decoupling the sub-component objects from the sub-component prerequisite objects, reusability of the page sub-component objects 450 is greatly enhanced.

[0052] The page sub-component prerequisite objects 460 ensure that the conditions for which they are responsible are met. Typically, page sub-component prerequisite objects 460 display information to the user and/or collect some information from the user and ensure the correctness of the collected information. In the embodiment depicted, the page sub-component prerequisite objects 460 store and retrieve the state of their requirements to/from the status data store 470. For example, once a required disclosure is displayed to the user that is a precondition to display of a particular page sub-component object, the associated page sub-component prerequisite object may record that fact in the status data store 470. Then, subsequently, when the particular page sub-

component object is requested, its output will be seen rather than that of its page sub-component prerequisite object.

Page Sub-Component Display Processing

5 **[0053]** **Figure 5A** is a flow diagram illustrating web page sub-component display processing according to one embodiment of the present invention. In one embodiment, the processing blocks described below may be performed under the control of a programmed processor, such as processor 202. However, in alternative embodiments, the processing blocks may be fully or partially implemented by any programmable or
10 hardcoded logic, such as Field Programmable Gate Arrays (FPGAs), TTL logic, or Application Specific Integrated Circuits (ASICs), for example.

[0054] In the example illustrated, display processing begins at processing block 500 where a request for a destination page, such as an HTTP request, is received by the web server 305 and conveyed to the object application 320 by way of the application
15 server 310.

[0055] At processing block 510, the sub-components of the destination page are identified. According to one embodiment, this is accomplished by the container object 420 querying the properties data store 440. In alternative embodiments, the sub-component prerequisite factory 430 may query the properties data store 440.

20 **[0056]** At processing block 520, the page sub-component prerequisites for the destination page sub-components are identified. As above, depending upon the implementation, this may be accomplished by the sub-component prerequisite factory 430 querying the properties data store 440. In alternative embodiments, the container object 420 may query the properties data store 440. Advantageously, by defining the
25 relationships between prerequisites and destination pages in a data store, no compilation is required when the system administrator modifies those relationships. Rather, if the data store is a text file, it may simply be modified by an editor.

[0057] At processing block 530, an appropriate sub-component object (either a page sub-component objects or a page sub-component prerequisite object 460) is added to the destination page's list of sub-component objects to display for each sub-component identified in processing block 510. According to one embodiment, the container object 420 calls the "getNextPreReq" method for each of the page sub-components of the destination page and adds the either the object returned (a page sub-component prerequisite object) or the page sub-component object to the destination page's list for subsequent display. According to one embodiment, the "getNextPreReq" method sequentially checks page sub-component prerequisites of the specified page sub-component in an order indicated by the properties data store and returns the first page sub-component prerequisite that has one or more requirements that are not satisfied. Advantageously, in this manner, hierarchical relationships of prerequisites may be defined easily.

[0058] Finally, at processing block 540, the destination page is displayed with the appropriate page sub-component outputs and appropriate page sub-component prerequisite outputs by causing each of the sub-component objects on the destination page's list to stream its output to the base agent 410.

[0059] **Figure 5B** is a flow diagram illustrating processing block 530 of **Figure 5A** according to one embodiment of the present invention. In this example, the goal is to find the first page sub-component prerequisite object associated with the specified page sub-component object that has one or more unmet requirements. At processing block 531, an instance of the next page sub-component prerequisite object is created by the sub-component prerequisite factory 430. Then, the sub-component prerequisite factory 430 determines if the page sub-component prerequisite object's requirements have been satisfied, at decision block 532, by invoking a method of the newly created instance, such as a "metReqs" method that all page sub-component prerequisite objects may implement.

If all the page sub-component prerequisite object's requirements have been met, then flow continues with decision block 534; otherwise processing branches to processing block 533.

[0060] At processing block 533, since it has been determined that one or more requirements of the page sub-component prerequisite object has not been met, the instance of the page sub-component prerequisite object is returned by the sub-component prerequisite factory 430 and added to the destination page's list. In this example, after the first page sub-component prerequisite object has been found with one or more unsatisfied requirements (i.e., after processing block 533), control returns to processing block 540.

[0061] At decision block 534, a determination is made whether or not there are more page sub-component prerequisites. If so, processing continues with processing block 531. If not, no page sub-component prerequisite objects have been found that have unmet requirements; and processing continues with processing block 535. At processing block 535, because either the specified page sub-component has no page sub-component prerequisites or all have been satisfied, an instance of the specified page sub-component is added to the destination page's list.

Architecture of another Exemplary Object Application

[0062] **Figure 6** is a block diagram of a software architecture for an object application according to another embodiment of the present invention. The architecture depicted seeks to facilitate the management, tracking and controlling of page prerequisites. According to this example, the software architecture of the object application 320 includes a base agent 610, a prerequisite factory 630, page objects 640, page prerequisite objects 650, a properties data store 630, and a status data store 660.

[0063] The base agent 610 includes logic to determine which page object is responsible for streaming out the content of the destination page based upon predetermined properties stored in the properties data store 630 and/or the specified URL.

According to one embodiment, when the user requests the destination page by selecting a link or specifying a URL, for example, in the client browser, the request is passed to the application server 310 via the web server 305. Then, the application server 310 determines the appropriate base agent 610 for the specified URL, creates an instance of the base agent 610, and calls the execute method of the base agent 610. After the base agent 610 has determined the appropriate page object associated with the URL, it consults with the prerequisite factory 620 to determine what the next prerequisite is for the page object. If there are no prerequisites or all the prerequisites have been satisfied then the base agent 610 creates an instance of the page object and executes it to cause the object to stream its output to the display. However, if a prerequisite is returned by the prerequisite factory 620, then the base agent 610 instead executes the page prerequisite object causing the prerequisite object to stream its output to the display.

[0064] According to one embodiment, the prerequisite factory 620 includes a method called “getNextPreReq” that returns either the next page prerequisite object for a specified destination page or null. First, the prerequisite factory 620 determines what, if any, page prerequisite objects 650 have been associated with the specified destination page. In the example depicted, this operation is accomplished by retrieving information from the properties data store 630, such as the value of a “prereq” property. Second, for each of the page prerequisite objects identified, the prerequisite factory 620 determines if the requirements for the page prerequisite object have been satisfied. As described further below, in one embodiment, this determination is made by calling a “metReqs” method of a “decider” component of the page prerequisite object. Because pages may have multiple sub-components, it is preferable to designate one as a decider component that can be used to determine whether or not the conditions for which it is responsible have occurred. In any event, if the prerequisite factory 620 returns a non-null object, it means the destination page is associated with a prerequisite managed by the page prerequisite object that has not yet been satisfied; and rather than displaying the

destination page, the page prerequisite object should be displayed to the user. If a null value is returned by the prerequisite factory 620, it means that either the specified destination page has no prerequisites or that all the prerequisites for the destination page have been fulfilled and therefore the destination page may be displayed. Advantageously, according to this novel architecture, the base agent 610 is abstracted from the task of prerequisite processing. In alternative embodiments, the base agent 610 may be even further abstracted from this process by removing its interface with the properties data store 440 and requiring all access to the properties data store 630 be by way of the prerequisite factory 620.

10 [0065] As above, the page objects 640 and the page prerequisite objects 650 may be coded independently of each other and do not even need to be aware of each other. Thus, reusability is enhanced because the page objects 640 and the page prerequisite objects 650 are decoupled thereby allowing page objects 640 to be easily taken out of the current context and placed unchanged into a new context.

15 [0066] The page prerequisite objects 650 ensure that the conditions for which they are responsible are met. Typically, page prerequisite objects 650 display information to the user and/or collect some information from the user and ensure the correctness of the collected information. In the embodiment depicted, the page prerequisite objects 650 store and retrieve the state of their requirements to/from the status data store 660. For example, once a required disclosure is displayed to the user that is a precondition to display of a particular page object, the associated page prerequisite object may record that fact in the status data store 660. Then, subsequently, when the particular page object is requested, its output will be seen rather than that of its page prerequisite object.

20 [0067] Importantly, the exemplary software architectures depicted in **Figures 4 and 6** are not mutually exclusive. Rather, it is anticipated that depending upon the particular implementation either or both page sub-component prerequisites and page

prerequisites may be used in the context of the same user interface for purposes of achieving the desired objectives.

Page Display Processing

5 **[0068]** **Figure 7A** is a flow diagram illustrating web page display processing according to one embodiment of the present invention. In one embodiment, the processing blocks described below may be performed under the control of a programmed processor, such as processor 202. However, in alternative embodiments, the processing blocks may be fully or partially implemented by any programmable or hardcoded logic, 10 such as Field Programmable Gate Arrays (FPGAs), TTL logic, or Application Specific Integrated Circuits (ASICs), for example.

15 **[0069]** In the example illustrated, display processing begins at processing block 710 where a request for a destination page, such as an HTTP request, is received by the web server 305 and conveyed to the object application 320 by way of the application server 310.

20 **[0070]** At processing block 720, the page prerequisite objects for the destination page are identified. According to one embodiment, this is accomplished by the base agent 610 calling the “getNextPreReq” method of the prerequisite factory 620, which, in turn, queries the properties data store 630. Again, the advantage of defining relationships between prerequisites and destination pages in a data store is the fact that no compilation is required when the system administrator modifies those relationships.

25 **[0071]** At processing block 730, prerequisite page objects that have one or more requirements that are not fulfilled are displayed sequentially until all prerequisite page objects’ requirements are satisfied. According to one embodiment, the “getNextPreReq” method sequentially checks page prerequisite objects associated with the destination page in an order indicated by the properties data store 630 and returns the first page prerequisite object that has one or more requirements that are not satisfied thereby

facilitating definition of hierarchical relationships among page prerequisite objects and page objects.

[0072] Finally, after all the page prerequisite objects' requirements have been satisfied at processing block 740, the destination page is displayed.

5

[0073] **Figure 7B** is a flow diagram illustrating processing block 730 of **Figure 7A** according to one embodiment of the present invention. In this example, the goal is to find the first page prerequisite object associated with the specified page object that has one or more unmet requirements. At processing block 731, the "decider" sub-component of the next page prerequisite object is determined. Again, the decider is the sub-component of the page prerequisite object that is responsible for communicating whether or not the page prerequisite object's requirements have been met. According to one embodiment, the decider sub-component is determined by the prerequisite factory 620 querying the properties data store 630 for the "decider" property of the page prerequisite object.

10

15

[0074] At processing block 732, an instance of the "decider" sub-component is created. According to one embodiment, a special method is made available to allow creation of the page prerequisite object for the purpose of checking if the requirements have been met. Alternatively, the standard create method may be called with a control flag indicating the purpose of the create call, e.g., check prerequisites vs. true creation. At any rate, at decision block 733, the prerequisite factory 620 determines if the page prerequisite object's requirements have been satisfied by invoking a method of the newly created instance, such as a "metReqs" method that preferably all "decider" sub-components implement. If all the page prerequisite object's requirements have been met, then flow continues with decision block 735; otherwise processing branches to processing block 734.

20

25

[0075] At decision block 735, a determination is made whether or not there are more page prerequisite objects associated with the destination page. If so, processing continues with processing block 731. If not, no page prerequisite objects have been found that have unmet requirements; and processing continues with processing block 740.

5 [0076] At processing block 734, since it has been determined that one or more requirements of the page prerequisite object has not been met, the output of the prerequisite page is streamed to the display. Therefore, in this example, after the first page prerequisite object has been found with one or more unsatisfied requirements that page is displayed rather than the destination page selected by the user. In one
10 embodiment, because the decider sub-component has already been instantiated, an optimization involves the framework saving the decider object created during the metReqs call so that it does not have to be recreated when the prerequisite page object is executed.

[0077] At decision block 736, a determination is made whether or not the page
15 prerequisite has been closed. If so, control branches to decision block 733. If not, control loops back to decision block 736 thereby preventing the destination page from being displayed until the requirements of the prerequisite page object have been fulfilled.

Properties File Syntax Enhancements

20 [0078] Before describing various improvements to existing properties file syntax, it may be instructive to briefly discuss the current usage model of such files with reference to **Figure 8**. As discussed above, Java has a special built-in mechanism for handling internationalization. Locale specific properties/attributes are placed in files having a “.properties” extension (herein referred to as a “properties file”). The properties
25 file contains one or more lines of attribute name value pairs (hereinafter “attribute-value pairs”) in the general form represented by expression 810. The first portion of the attribute-value is an attribute name, such as “prompt1”. The attribute name is followed

by an equal sign. The final portion of the attribute value pair comprises a string value, such as “please enter your name”, to be associated with the particular attribute. In the Java framework, properties files may be loaded into objects of type “ResourceBundle”.

Then, the resulting resource bundle objects may be queried to determine a value associated with a particular attribute of interest. For example, if a properties file contained the line:

Abc.def.gh_I=j

then invoking the appropriate resource bundle object’s get method, e.g.,

`resourceBundle.getString(“Abc.def.gh_I”) would return the string value “j”.` In this

manner, applications can query the value of an attribute during run time. However, one limitation of this prior art syntax is that both properties files and the corresponding internal resource bundle representations are flat, non-object oriented collections of key-value pairs. In connection with an object-oriented implementation of page sub-component prerequisites, page prerequisites, and/or other applications, the assignee of the present invention has found it to be advantageous to (1) allow properties/attributes to be defined and accessed on an object-by-object basis; (2) enable inheritance of properties/attributes through some hierarchy of objects; (3) specify and override properties/attributes of objects as they are used in different contexts; and (4) enable inheritance of contexts.

Definition and Access of Properties on an Object-by-Object Basis

[0079] Figure 9 illustrates an improved properties file syntax that enables definition and access of properties on an object-by-object basis according to one embodiment of the present invention. According to the embodiment depicted, a properties data store, such as a properties file, may include attribute-value pairs according to the general form represented by expression 910. The first portion of the attribute-value pair is a string value representing an attribute name, such as “prompt1”. Expression 910

further includes a string value representing an object name, followed by a delimiter, such as ‘.’. As above, the attribute name is followed by an equal sign. Finally, a value to be associated with the particular object attribute is specified in the form of a string value.

While preferably, for readability, the delimiter that separates the object name and the

attribute name is a non-alphanumeric printable ASCII character, in alternative embodiments other single or multiple character delimiters may be employed.

[0080] The syntax represented by expression 910 allows one or more attribute-value pairs to represent what is referred to herein as a “properties file object” or simply an object. For example, attribute-value pairs 920 and 930 together define a properties file object called “*Obj1*” having two attributes, *attr1* and *attr2*. Similarly, attribute-value pairs 940 and 950 specify a properties file object named “*Obj2*” that also includes two attributes, *attr1* and *attr3*. Attribute-value pair 920 associates the value, *val11*, with the attribute named *attr1* of object, *Obj1*. Likewise, attribute-value pairs 930, 940, 950, and 960, associate *val12*, *val21*, *val23*, and “please enter your name” with *attr2*, *attr1*, *attr3*, and *prompt1* of *Obj1*, *Obj2*, *Obj2*, and *Trading*, respectively.

[0081] Importantly, this improved syntax is fully compliant with standard Java properties files thereby making this syntax enhancement transparent to Java. To Java, the attribute-value pairs 920, 930, 940, and 950 have no additional meaning. Rather, Java will interpret them as simply representing four independent attribute-value pairs. For example, Java interprets the attribute-value pair 920 as the value, *val11*, being associated with an attribute named “*Obj1.attr1*”. Similarly, attribute-value pairs 930, 940 and 950, respectively, are interpreted as the value, *val12*, being assigned to “*Obj1.attr2*”, the value, *val21*, being assigned to an attribute named “*Obj2.attr1*”, and “*Obj2.attr3*” being associated with a value of *val23*. Consequently, as described further below, the syntax enhancements discussed herein may be implemented on top of standard Java properties file semantics by supplying the standard Java Application Programming Interfaces (APIs)

and/or resource bundle methods with attribute names that include appropriate delimiters to separate the object name from the attribute name.

Inheritance of Properties

5 **[0082]** Currently, inheritance is not supported by the flat, non-object oriented structure of properties files. Therefore, the concept of a base object (e.g., an object upon which other objects may be built) does not exist. If common attributes are required in a properties file, they must be replicated as many times as necessary. Unfortunately, such replication becomes problematic when changes become necessary to the common
10 attributes since each change must be accurately performed for each of the replicated attributes.

[0083] **Figure 10** illustrates an improved properties file syntax that enables inheritance of properties according to one embodiment of the present invention. In this example, a special attribute-value pair having the form of expression 1010 is provided. A
15 first portion of the special attribute-value pair representing an object name is separated from a special attribute name, “extends”, by a delimiter. The “*extends*” attribute is used to specify inheritance from the object specified on the right hand side of the equal sign.

[0084] As illustrated by attribute-value pairs 1020, 1030, 1040, 1050, and 1060, this mechanism allows attributes and values of objects to be inherited by other objects;
20 and the other objects may each override the value of an inherited attribute as well as add its own attributes and values. In the present example, two “properties file objects” are defined, *Obj1* and *Obj2*. *Obj1* and its associated attributes and values are defined by attribute-value pairs 1020 and 1030. *Obj2* and its associated attributes and values are defined by attribute-value pairs 1040, 1050, and 1060. The use of the special attribute
25 name “*extends*” in attribute-value pair 1040 causes *Obj2* to extend *Obj1*. That is, *Obj2* inherits all the attributes of *Obj1*. Additionally, in this example, attribute-value pair 1050 overrides the value of *attr2* of *Obj2* with *val22*. Finally, attribute-value pair 1060

specifies that *Obj2* has an attribute, *attr3*, in addition to those specified for *Obj1*, which is assigned the value *val23*.

[0085] Advantageously, this novel inheritance syntax saves time for the person specifying attributes. For example, in a system in which two or more objects have many attributes and values in common, the common attributes need only be specified once in a base object. Then, extending objects need only specify overridden values or new attributes and values. Thus, by employing the inheritance mechanism described herein, time savings proportionate to the commonality of attributes among objects of a system can be achieved during specification of the attributes.

[0086] This notation also facilitates maintainability of properties files in which it is employed. Rather than replicating common attributes among many objects as would be required in a non-hierarchical environment that lacks a mechanism for inheritance, such as that imposed by current properties file syntax, the notation of **Figure 11** facilitates modification of object attributes and/or attribute values by allowing common attributes to be specified in base objects. As a result, a change to a particular common attribute is localized to the base object that specifies that common attribute; and semantically this change is automatically accomplished for all sub-objects (e.g., extending objects) that inherit from the modified base object.

[0087] While the special attribute name “*extends*” has been chosen to enhance the readability and understandability of the properties file, the invention is not limited to any particular naming convention. Thus, in alternative embodiments, the name “*extends*” may be replaced with other names, such as “*inheritsfrom*”, “*isa*”, or other arbitrary string values.

Specification and/or Overriding of Object Properties in Different Contexts

[0088] **Figure 11** illustrates an improved properties file syntax that enables the specification and/or overriding of object properties as they are used in different contexts

according to one embodiment of the present invention. In this example, the syntax of an attribute-value pair may include a context name, an object name, and an attribute name as illustrated by expression 1110. The context may represent a current state of a system, a top level container, a sub-container, a page, or other objects or circumstances useful for interpreting a particular object. In the examples described herein, the context is typically a user interface screen, such as a web page.

[0089] Importantly, this enhanced syntax provides for context sensitive overriding of properties file object attributes. In the present case, attribute-value pairs 1120 and 1130 define an object, *Obj1*, having *attr1* and *attr2*. The value of *attr1* is always *val11*.

However, as a result of the attribute-value pair 1140, the value of *attr2* is *val12* except when *Obj1* is running in the context of *Page1* in which case the value of *attr2* is *val112*. Therefore, assuming an object has the ability to determine its context, a mechanism is provided by the syntax of expression 1110 for context sensitive overriding of attributes of objects.

[0090] While, in the examples illustrated herein, the context name, object name, and attribute name appear in a specific order relative to each other within attribute-value pairs, the invention is not limited to any particular ordering of these names.

Consequently, in other embodiments, alternative orderings of these names may be used.

Inheritance of Contexts

[0091] **Figure 12** illustrates an improved properties file syntax that enables inheritance of contexts according to one embodiment of the present invention. The concept enabled by this syntax enhancement is similar to that of **Figure 10** with reference to which inheritance of attributes of properties file objects was illustrated. However, here, the properties file object is a context. Since a context is just a properties file object, any context may be extended and any contexts' attributes may be overridden or added as discussed earlier.

[0092] In the present example, properties files may include attribute-value pairs having the form of expression 1210, which includes a context name, a special attribute name, “*extends*”, and an indication of the context to be extended (e.g., the context from which to inherit objects and/or attributes). Attribute-value pairs 1220 and 1230 indicate

5 that normally, or by default, the value of attribute *attr2* of *Obj1* is *val12*. However, attribute-value pair 1240 overrides the definition of *attr2* of *Obj1* when *Obj1* is running in the context of *Page1*. Attribute-value pairs 1250 and 1260 specify a context, *Page2*, that includes all the objects and attributes of *Page1* but overrides the value of *attr2* of *Obj1* when *Obj1* is running in the context of *Page2*. Thus, the value of *attr2* of *Obj1* may
10 have one of three values depending upon its current context. For example, when the context is *Page1*, *attr2* is *val112*; when the context is *Page2*, *attr2* is *val212*; otherwise, *attr2* is *val12*.

Property Retrieval Processing

[0093] **Figure 13** is a high-level flow diagram that conceptually illustrates property retrieval processing according to one embodiment of the present invention. For simplicity the present example has been broken down into three stages: (1) a first stage in which the object hierarchy is traversed for a fixed context, (2) a second stage in which both context hierarchy traversal and object hierarchy traversal are performed, and (3) a third stage in which context is disregarded during object hierarchy traversal.

[0094] In the embodiment depicted, the property retrieval process begins at processing block 1310. Processing block 1310 is representative of the first stage of the three stages. During processing block 1310, an attempt is made to retrieve a specified attribute by climbing a specified object hierarchy (presumably created by the inheritance mechanisms described earlier) with which the attribute is thought to be associated. If a value for the specified attribute is located during this stage, processing is complete; otherwise processing proceeds with processing block 1320 (the second stage). At processing block 1320, the context is no longer held fixed as in processing block 1310. Rather, the context hierarchy is climbed to search parent contexts. Within each context, every level of the specified object hierarchy is also searched until the desired value is located or the levels are exhausted. Again, if the desired value is found, processing may cease at this point; otherwise, a final attempt is made to retrieve the value of the specified attribute in processing block 1330. During processing block 1330 (the third stage), context is ignored; and a final search is made for the desired value by climbing the object hierarchy without regard to the current context.

[0095] **Figure 14A** is an exemplary context and object hierarchy. In this example, the properties file defines the following object hierarchy: *Obj3* extends *Obj2* which extends *Obj1*, and the following context hierarchy *Page3* extends *Page2* which extends *Page1*.

[0096] **Figure 14B** illustrates property retrieval processing according to the object and context hierarchy of **Figure 14A** and the processing of **Figure 13**. Suppose an object of type *Obj3* running in the context of *Page3* wants to get the value of its attribute, *attr*. According to the search approach described with reference to **Figure 13**, the object will look for a value of *attr* in the order shown in **Figure 14B**. The retrieval process will stop as soon as one of the requested attributes is located in the properties file or resource bundle representation.

[0097] First, the object will attempt to locate the value of *attr* by searching a first group of attribute-value pairs 1410. More specifically, the object will first look in its current context, e.g., by calling its *getValue()* method with the string value “*Page3.Obj3.attr*”. If the value of *attr* for *Obj3* is not overridden by *Page3*, then the object will begin climbing its hierarchy. In this example, the object will look to its parent, *Obj2*, to see if a value has been associated with *attr*, e.g., by making the call *getValue(“Page3.Obj2.attr”)*. If this retrieval is unsuccessful, e.g., returns a null value, then a final attempt within the current context will be made to retrieve the desired value from *Obj1*.

[0098] Assuming the desired value was not found during the first stage. The search continues into a second group of attribute-value pairs 1420. The search of the second group 1420 represents the second stage where an attempt is made to locate the object’s attribute by climbing the object’s hierarchy in the parent context, *Page2*, until a level is found in which the attribute is assigned a value or until the levels are exhausted. The search proceeds as above, climbing the object hierarchy within the current context, *Page2*. Once the object hierarchy has been exhausted, however, the search continues with the parent of the current context, *Page1*.

[0099] Again, assuming the desired value was not located by a previous stage, the search enters a final stage in which context is disregarded and the object hierarchy is traversed. First, the object determines if it has defined a value for *attr*, e.g., by making

the call *getValue("Obj3.attr")*. If this call is unsuccessful, then a subsequent retrieval is made with reference to the object's parent, *Obj2*. If that call fails, a final attempt to retrieve the desired value will be made with reference to *Obj2's* parent, *Obj1*, e.g., by calling *getValue("Obj1.attr")*.

5

[0100] **Figure 15** is a flow diagram illustrating processing associated with traversal of an object hierarchy and a context hierarchy according to one embodiment of the present invention. According to this embodiment, the first and second stages of the search processing are combined into a single stage. At processing block 1510, an initial attempt is made to retrieve the value of the requested attribute from the current object while taking into consideration the current context. At decision block 1520, a determination is made whether or not the attribute has been found. If not processing continues with decision block 1530; otherwise the search is complete.

10

[0101] At decision block 1530, a determination is made whether or not the object hierarchy has been exhausted. If the current object has a parent, then another level in the object hierarchy exists. However, if the current object does not have a parent, then the object hierarchy has been exhausted. At any rate, assuming the object hierarchy has been exhausted, processing continues with processing block 1540 during which the current object is updated to be the next object in the object hierarchy, i.e. its parent, and processing returns to processing block 1510.

15

20

[0102] Returning to decision block 1530, if the object hierarchy has been exhausted, processing proceeds to decision block 1550. At decision block 1550 a determination is made whether or not the context hierarchy has been exhausted. If there is no parent of the current context (context hierarchy exhausted), then the current stage of the search is complete. Otherwise, if the current context has a parent (context hierarchy not exhausted), then processing continues with processing block 1560.

25

[0103] At processing block 1560, the current context is updated to the next context (i.e., the parent of the current context). Then, at processing block 1570, the current object is reset to the original object to restart object hierarchy traversal in the new context.

5 [0104] It is worth mentioning at this point that while the processing blocks of a flow diagram illustrating a particular embodiment are presented and described in a particular order, there should be no implication that the present invention is limited by that particular ordering. Rather, it is contemplated that in alternative embodiments, the ordering of the processing blocks may be changed while preserving the core functionality.

10 For example, in **Figure 15**, at least the ordering of processing blocks 1560 and 1570 may be reversed without any effect.

[0105] **Figure 16** is a flow diagram illustrating processing associated with traversal of an object hierarchy without regard to context according to one embodiment of the present invention. At processing block 1610, an initial attempt to retrieve the desired value is made by ignoring the current context. A determination is made regarding whether the retrieval was successful at decision block 1620. If the attribute value has successfully been located search processing is complete; otherwise, processing continues with decision block 1630. At decision block 1630, a determination is made whether or

15 not the object hierarchy is exhausted. If so, this third stage of processing is complete. Otherwise, processing proceeds to processing block 1640. At processing block 1640, the current object is updated to reflect the next object in the object hierarchy; and then processing returns to processing block 1610.

[0106] Importantly, while the flow diagrams described above imply a simple,

25 linear, non-recursive algorithm, it is contemplated that in an actual code implementation it may be desirable to implement one or more of the methods as recursive procedures/functions.

Software Architecture for Processing an Improved Properties File Syntax

[0107] **Figure 17** is a block diagram the conceptually illustrates software architectural components that enable processing of an improved properties file syntax according to one embodiment of the present invention. According to this example, an object 1710 exposes a *getValue()* method 1711 which in turn may call a *getContextInheritance()* method 1713 and/or a *getWithInheritance()* method 1717. In alternative embodiments, the *getValue()* method 1711 may be private (i.e., not exposed outside of the object 1710).

[0108] According to one embodiment, object 1710 inherits logic from a base object that serves as an abstraction for the interface to API 1720. In this manner, object 1710 automatically has the ability to retrieve the value of any of its attributes without needing knowledge of the underlying interface to the properties data store, e.g., properties file 1730. Additionally, inheritance of this logic saves design and development time.

[0109] In one embodiment, the functionality of the *getContextInheritance()* method and *getWithObjectInheritance()* method are represented by the processing described with reference to **Figure 15**; and the functionality of the *getWithInheritance()* method is represented by the processing described with reference to **Figure 16**.

[0110] Importantly, while, for sake of generality, the properties file 1730 is shown as being accessed by API 1720, it is appreciated that the properties file 1730 may be internally represented with a *resourceBundle* object as described above. Also, if this is the case, the API 1720 would be the *resourceBundle*'s exposed methods.

Exemplary Methods Accessible by the Objects

[0111] For the sake brevity, in the code examples of **Figures 18-21**, it is assumed that there exists a *getContextname()* method in every object that returns a string value containing the context/container name. Every object also is assumed to know how to

figure out its name. In Java this is done by performing the call: *getClass().getName()*.

Subsequently, the left sub-parts of the name are removed to reveal the object name. In the following examples, this left sub-part removal (or alternatively, right sub-part extraction) is assumed to be performed by *getName()*. Therefore, “*pkg1.sub2.Obj3*”

5 would result in the object name of “*Obj3*”. In the exemplary implementation every object is additionally assumed to extend some base object that implements a *getValue()* method, which returns the value of a key-value pair corresponding to a specified key. Therefore, every sub-component inherits the logic in the method *getValue()*.

[0112] **Figure 18** illustrates exemplary code for the *getValue* method of **Figure**
10 **17** according to one embodiment of the present invention. In this example, the method returns the value of the requested attribute of the currently executing object, taking the current context into consideration.

[0113] **Figure 19** illustrates exemplary code for the *getContextInheritance*
15 method of **Figure 17** according to one embodiment of the present invention. In this example, this method looks for the value of attribute, *attrib*, of an object named *object* as it is running in the context, *context*. If the specified attribute is not found in the specified context, the method looks in the parent context. Importantly, as it looks for the attribute in a given context, it also examines the object’s hierarchy. The method uses recursion that terminates when (1) the desired value is found or (2) the whole hierarchy has been
20 examined and the specified attribute cannot be found.

[0114] **Figure 20** illustrates exemplary code for the *getWithObjInheritance*
method of **Figure 17** according to one embodiment of the present invention. In this example, this method looks for the value of attribute, *attrib*, of an object named *object* as it is running in the context, *context*. If the specified attribute is not found associated with
25 the object running in the current context, then the method looks for the attribute in the object’s parent running in the same context. That is, the method continues to climb up the object hierarchy while keeping the context fixed. The method uses recursion that

terminates when (1) the desired value is found or (2) the whole hierarchy has been examined and the specified attribute cannot be found.

[0115] **Figure 21** illustrates exemplary code for the *getWithInheritance* method of **Figure 17** according to one embodiment of the present invention. In this example, this method looks for the value of attribute, *attrib*, of an object named *object*. If the specified attribute is not found, then the method looks for the attribute in the object's parent and continues to climb the object hierarchy until the method terminates. The method uses recursion that terminates when (1) the desired value is found or (2) the whole hierarchy has been examined and the specified attribute cannot be found.

[0116] Importantly, while not shown in the exemplary code fragments, a memory cache may be implemented to avoid repeating lookups for attributes. Briefly, when an object needs to calculate the value of some property it first checks if it is available in the cache. If it is, the value is retrieved from the cache; otherwise, the value may be retrieved from the properties file (via the resourceBundle and algorithms described above or the exemplary code of **Figures 18-21**). If the value is found in the properties file, a copy may be stored in the cache to avoid future file lookups and related processing.

[0117] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.